Request and Response in PHP

How PHP Makes Complex Parsing Simple

- PHP Magic: Complex → Simple
- PHP Superglobals: Pre-parsed for You
  - $_SERVER - Request Information
  - $_GET - Query Parameters
  - $_POST - Form Data
- Parsing the Request in PHP
- Method Detection Made Simple
- Response Generation in PHP
  - Setting Headers
  - Sending JSON Response
- Routing Made Simple
- Error Handling Made Easy
- Before PHP vs With PHP
  - What PHP Handles Automatically
- PHP Superglobals Summary
  - Practical Example: Building Our API

PHP Magic: Complex → Simple

Remember the complex parsing from before?

PHP transforms this:

```
GET /api?name=John&age=25 HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0
...
```

Into this simple access:

```php
$method = $_SERVER['REQUEST_METHOD'];  // "GET"
$path = $_SERVER['REQUEST_URI'];       // "/api?name=John&age=25"
$name = $_GET['name'];                 // "John"
$age = $_GET['age'];                   // "25"
```

PHP Superglobals: Pre-parsed for You

- $_SERVER[...]
- $_GET[...]
- $_POST[...]

$_SERVER - Request Information

```php
$_SERVER['REQUEST_METHOD']  // GET, POST, PUT, DELETE
$_SERVER['REQUEST_URI']     // /api/users?id=123
$_SERVER['HTTP_HOST']       // localhost:8000
$_SERVER['HTTP_USER_AGENT'] // Browser information
$_SERVER['CONTENT_TYPE']    // application/json
```

$_GET - Query Parameters

From this HTML request line, PHP makes $_GET list.

```
GET /search?q=php&category=tutorial
```

We can use $_GET to get the information.

- q=php
- category=tutorial

```php
$_GET['q']        // "php"
$_GET['category'] // "tutorial"
```

$_POST - Form Data

```
// From HTML form or POST request body
$_POST['username'] // Form field value
$_POST['password'] // Form field value
```

- When users give information using an HTML form, we can use $_POST to get the information.

Parsing the Request in PHP

Extracting the path portion from a URL in PHP

- $_SERVER['REQUEST_URI'] contains the path information.
- It contains the whole URI (Unique Resource Information).
    - For example, from http://example.com/project-files0/index.php.php?a=10&b=20, we should get the path <project-files0/index.php>.

```php
// Get the request path - PHP makes this easy!
$path = $_SERVER['REQUEST_URI'];
$path = parse_url($path, PHP_URL_PATH);
$path = trim($path, '/');
```

From the input http://example.com/abc?a=b&c=d

- `$_SERVER['REQUEST_URI']` => `/abc?a=b&c=d`
- `parse_url($path, PHP_URL_PATH)` => `/abc`
- `trim($path, '/')` → `abc`

```php
$path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
```

The `parse_url()` function Parses a URL and returns its components.

- Extracts only the path part from a full URI
- Removes query parameters, fragments, etc.
- Returns clean path for routing

What PHP does automatically:

1. ✅ Parses the entire HTTP request
2. ✅ Populates `$_SERVER` with request data
3. ✅ Provides `parse_url()` function for URL parsing
4. ✅ Handles URL decoding automatically

Method Detection Made Simple

- The request method can be GET, POST, DELETE, or any other.
- PHP $_SERVER['_REQUEST_METHOD'] contains the request method.

Raw HTTP: You'd need to parse `GET /api HTTP/1.1`

PHP: One line!

```php
if ($_SERVER['REQUEST_METHOD'] !== 'GET') {
    sendError('Only GET requests are allowed', 405);
    exit;
}
```

What this replaces:

```python
# Manual parsing (hundreds of lines)
def parse_request_method(raw_request):
    first_line = raw_request.split('\r\n')[0]
    method = first_line.split(' ')[0]
    return method
```

Response Generation in PHP

- From the requests, the web server processes information.
- Finally, the web server generates a response.
  - In this example, we generate a header and content.
  - The content can be anything, but in this example, we use JSON.
  - The header should specify that the content is JSON.

Setting Headers

```php
// Set content type as JSON
header('Content-Type: application/json');

// Set status code
http_response_code(404);  // Not Found
http_response_code(200);  // OK
```

Sending JSON Response

```php
function sendResponse($data, $message = 'Success') {
    echo json_encode([
        'success' => true,
        'message' => $message,
        'data' => $data
    ], JSON_PRETTY_PRINT);
}
```

- json_encode() function transforms a PHP dictionary into a JSON string.

PHP handles:

- ✅ JSON encoding
- ✅ Character encoding
- ✅ Content-Length header
- ✅ Proper HTTP formatting

Routing Made Simple

```
$path = $_SERVER['REQUEST_URI'];
$path = parse_url($path, PHP_URL_PATH);
$path = trim($path, '/');
```

- In this code, $path has the path part of the input: abc from http://example.com/abc?a=b&c=d.
- We can use `$path` to determine how to route.

Our example shows clean routing:

```php
switch ($path) {
    case '':
    case 'api':
        $info = [
            'name' => 'Simple PHP API for Education',
            'version' => '1.0',
            'endpoints' => [
                'GET /' => 'Show this API information'
            ]
        ];
        sendResponse($info, 'Welcome to Simple PHP API');
        break;

    default:
        sendError('Endpoint not found', 404);
        break;
}
```

What you get for free:

- Clean URL handling
- Simple pattern matching
- Easy response generation

Error Handling Made Easy

Our example:

```php
function sendError($message, $code = 400) {
    http_response_code($code);
    echo json_encode([
        'success' => false,
        'message' => $message,
        'data' => null
    ], JSON_PRETTY_PRINT);
}
```

We can use sendError function to respond with 'Not found' message of 404 code.

```
sendError('Not found', 404);
```

Or 'Bad request' of 400 code.

```
sendError('Bad request', 400);
```

Manual equivalent would require:

- HTTP status line formatting
- Header formatting
- JSON encoding
- Error handling
- Content-Length calculation

Before PHP vs With PHP

- Before PHP (Manual Parsing)

```
# Hundreds of lines like this:
raw_request = "GET /api?name=John HTTP/1.1\r\nHost: localhost\r\n\r\n"
lines = raw_request.split('\r\n')
first_line = lines[0].split(' ')
method = first_line[0]  # "GET"
url_with_query = first_line[1]  # "/api?name=John"
path, query = url_with_query.split('?', 1) if '?' in url_with_query else (url_with_query, '')
# ... more complex parsing
```

- With PHP

```
$method = $_SERVER['REQUEST_METHOD'];  // "GET"
$path = $_SERVER['REQUEST_URI'];       // "/api?name=John"
$name = $_GET['name'];                 // "John"
```

What PHP Handles Automatically

1. **Request Parsing**

- HTTP method detection
- URL parsing and decoding
- Header parsing
- Query parameter extraction
- POST data parsing

2. **Security**

- Input sanitization helpers
- SQL injection prevention (with prepared statements)
- XSS protection helpers

3. **Response Generation**

- Header management
- Status codes
- Content encoding
- Output buffering

PHP Superglobals Summary

Superglobal	Contains	Example
`$_GET`	Query parameters	`$_GET['id']`
`$_POST`	Form/POST data	`$_POST['username']`
`$_SERVER`	Server/request info	`$_SERVER['REQUEST_METHOD']`
`$_COOKIE`	Cookies	`$_COOKIE['session_id']`
`$_FILES`	Uploaded files	`$_FILES['upload']['name']`
`$_SESSION`	Session data	`$_SESSION['user_id']`

All automatically parsed and ready to use!

Practical Example: Building Our API

```php
// 1. Check request method (one line!)
if ($_SERVER['REQUEST_METHOD'] !== 'GET') {
    sendError('Only GET requests are allowed', 405);
    exit;
}

// 2. Get the path (built-in functions!)
$path = $_SERVER['REQUEST_URI'];
$path = parse_url($path, PHP_URL_PATH);

// 3. Route the request (simple switch!)
switch ($path) {
    case '/api':
        sendResponse($data);
        break;
    default:
        sendError('Not found', 404);
}
```


This replaces hundreds of lines of manual parsing!

Five key benefits of using PHP

- Why PHP Makes Web Development Easy

1. **Superglobals** provide instant access to request data
2. **Built-in functions** handle complex parsing
3. **Automatic security** features protect against common attacks
4. **Simple syntax** makes code readable and maintainable
5. **Rich ecosystem** of functions for web development

PHP Magic

PHP turns complex HTTP protocol handling into simple variable access!